ANALOG DEVICES

Accelerator Architecture & Micro-architecture definition

Rev: 0.01

Date: April 7, 2000~~ February 29, 2000~~

File: <file allocation>

History

Written by: Zvi G	Date: December 21, 1999
Modified By: Zvi G	Date: February 22, 2000
Modified By: Zvi G	Date: February 29, 2000

1. Introduction

The accelerator is targeted for Viterbi and Turbo-code algorithms, which are the major part of base-station applications. The accelerator executes new compute block instructions that simplify the coding of the two algorithms.

The advantage of supporting the Viterbi and Turbo-code in software is the flexibility of the support. When the algorithm is written in software using the compute block registers and accelerator registers it is simple to tune the algorithm according to specific requirements of the user.

The major strength of the TigerSharc is the huge data transfer rate – two 128-bit memory accesses every cycle. This enables calculation of a whole 8 state trellis calculation every two cycles in every compute block.

1.1. Data Types and Sizes

The accelerator instructions are targeted for implementing Viterbi and Turbo-code algorithms – mostly used in cellular applications. The input from the analog front end is normally up to 8 bits, and the range is of –1 to +1. Therefore the binary point is assumed to be between bits 7 and 8.

Turbo-code algorithm sums three numbers per sample point in the whole block. In 384K bit / sec standard the block is 2K size. The full result is a sum of 6K numbers of 8 bit wide. In order to represent the worst case final result we need 21 bits. For 2Mbit / sec the data block is 10K sample points i.e. 30K numbers. This requires 24 bits. In order to avoid overflow the data elements are 32 bit wide.

For Viterbi algorithm the number of sums is smaller – up to 2K numbers. In some cases (if analog input is up to 5 bits) 16 bit support is enough. On the other hand if the number of states is more than 16 (normally it is 64 to 256) the metrics of each state should be saved in memory and restored later. This may double the data transfer throughput (relative to Turbo-code algorithm). Implementing the algorithm in 16 bit format will ease the data throughput transfer. In short operation there is no meaning to the binary point place because TMAX function is not supported in 16 bit operations.

1.2. Tmax Function

The TMAX function is commonly used in Turbo-code. The function is:

TMAX (A, B) = max (A, B) + ln (1 + e ** (-lA – Bl))

The second part - ln (1 + e ** (-lA – Bl)) is implemented as a table. Since the function of the table is in the range of ~0.7 (if A = B the output is ln of 2) to zero (for large lA – Bl, ln of 1 is 0).

1.3. Trellis function

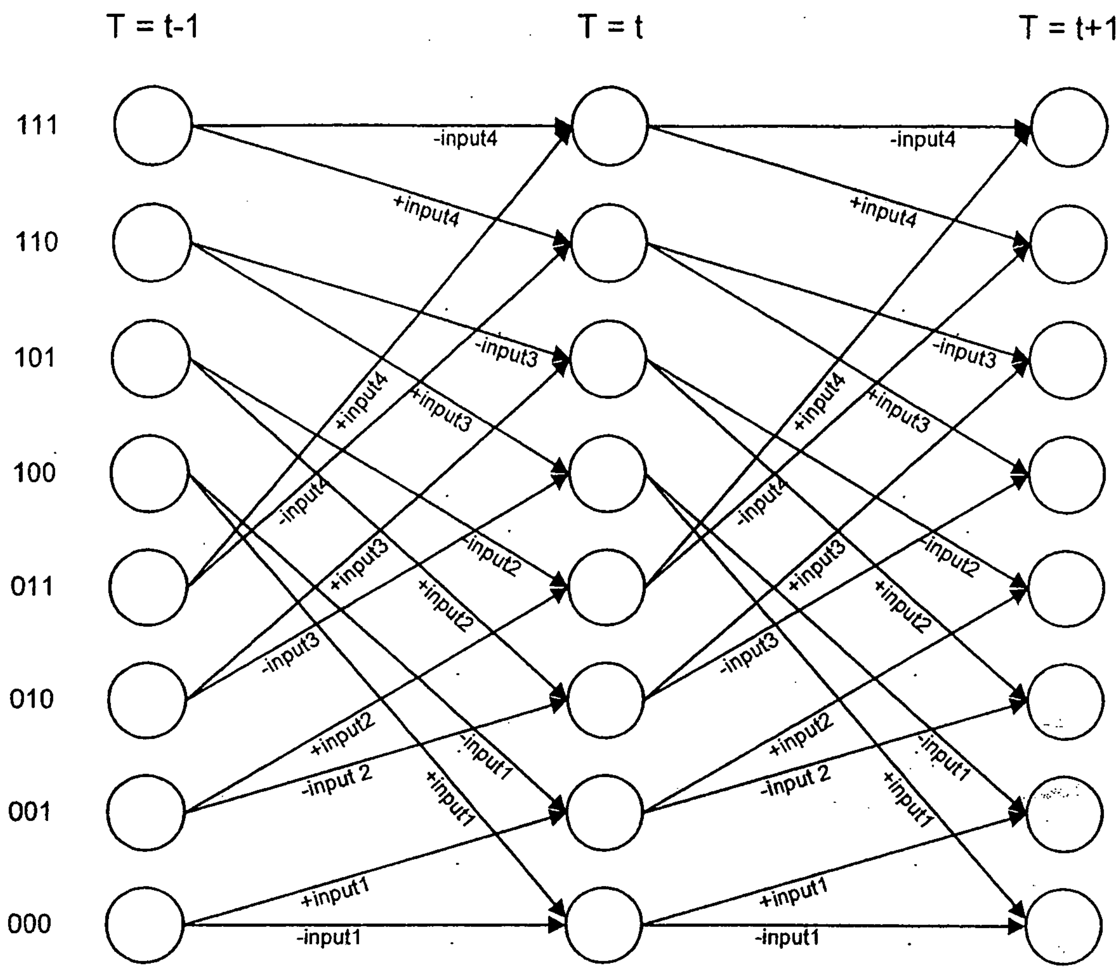
The Trellis function is widely used in both Viterbi and Turbo-code algorithms. The algorithm is based on Trellis diagram shown below.

In each state and each time-point we need to calculate the *metrics* of the point. The metrics of a point is calculated by the metrics of the state that came from added or subtracted with the input value (according to +input or –input on the arrow). Each new state has two results according to two possible previous states. The trellis function selects the maximum between the two possible values. For example the new value of state 100 at time t is

Metrics (100, t) = max ((Metrics (010, t-1) – input3), (Metrics (110, t-1) + input3))

Viterbi algorithm requires keeping the path that was selected, and the final metrics (metrics of the last time unit in block). Turbo-code requires keeping all metrics in the block. Also in turbo-code the MAX function is replaced to TMAX function (see above).

The instruction used for trellis function is ACS – Add Compare and Select. The function uses internal accelerator registers for keeping the metrics of points. The input is a compute register, and the result is written into the accelerator registers.


T = t-1
T = t
T = t+1
111
110
101
100
011
010
001
000
-input4
+input4
-input3
+input3
-input2
+input2
-input1
+input1
+input4
-input4
+input3
-input3
+input2
-input 2
+input1
-input1

[redacted]

ANALOG DEVICES

2. Internal Registers

~~1.1.~~2.1. Data Registers

The accelerator uses 16 data registers TR0 through TR15, each register is 32 bit wide.

~~1.2.~~2.2. Trellis History Registers

Trellis history registers – THR0 ~~through~~ & THR1 – 32 bits each register
This register pair keeps the history of ACS selection decisions. On each ACS instruction execution THR1:0 are shifted right, and the selection vector (one bit for each ACS decision) is shifted into THR1:0 MSB's. The shift is by four in 32 bit operation and by eight in 16 bit operation.

~~1.3.~~2.3. Status Flags

Extra bits to X/YSTAT:
Bit #14: TROV – Trellis accelerator overflow
Bit #15: TRSPOV – Trellis accelerator sticky positive overflow
Bit #16: TRSNOV – Trellis accelerator sticky negative overflow

3. Accelerator Instructions

Instruction	25:20	19	18:17	16:15	14:11	10	9: 5	4:3	2:0
	CU & type	Op - Code							
Rs(d)(q) = TRm(d)(q)	01 1110	101		00	Rs(d)(q)		00000	NLQ**	TRm [3:1]
TRs(d)(q) = Rm(d)(q)	10 1110	10,TRsd[1]		TRsd [3:2]	00000		Rmd	NLQ**	000
Rs(d)(q) = THRm(d)(q)	01 1111	101		00	Rs(d)(q)		00000	NLQ**	THRm [3:1]
THRs(d)(q) = Rm(d)(q)	10 1111	10, THRsd[1]		TRsd [3:2]	00000		Rmd	NLQ**	000
TRsd = TMAX(TRmd + Rmq_h, TRnd + Rmq_l) ~~(init)~~	01 ~~111~~ 1110 ~~(?)~~	11,Trsd[1]		TRsd [3:2]	0000	TRnd [3]	Rmq	TRnd [2:1]	TRmd [3:1]
TRsd = TMAX(TRmd - Rmq_h, TRnd - Rmq_l) ~~(init)~~	01 ~~110~~ 1100 ~~(?)~~	11,Trsd[1]		TRsd [3:2]	0000	TRnd [3]	Rmq	TRnd [2:1]	TRmd [3:1]
(S)TRsq = ACS (TRmd, TRnd, Rm) (TMAX)	01 11S*T	100		TRsq [3:2]	0000	TRnd [3]	Rm	TRnd [2:1]	TRmd [3:1]
Rsq = TRaq, (S)TRsq = ACS (TRmd, TRnd, Rm) (TMAX)	01 11S*T	0	Traq [3:2]	TRsq [3:2]	Rsq [4:2],0	TRnd [3]	Rm	TRnd [2:1]	TRmd [3:1]
Rsd = Permute (Rm, -Rm, Rn)	01 1100	1	01	00	Rsd		Rm	Rn	
Rsd = Permute (Rmd, Rn)	01 1110	1	01	00	Rsd		Rmd	Rn	
Rs = X/Ystat	According to existing definition of instruction – accelerator refers only to bits 16:14								
X/Ystat = Rm	According to existing definition of instruction – accelerator refers only to bits 16:14								

* S: Short operation (if clear) or regular 32 bit operation (if set)
** NLQ: two bits that define the data size:

0X: single word, X is used as the LSB of TRm register in "Rs = TRm;" instruction, or LSB of TRs register in "TRs = Rm;" instruction. Same for THRm and THRs in trace history register load and store.
10: Long (64 bits)
11: Quad (128 bits)

~~1.1.~~3.1.Accelerator Registers Load - Store

3.1.1. Rs(d)(q) = TRm(d)(q)

The data in the register TRm is transferred to Rs. Data is single (32 bits), double (64 bits) or quad (128 bits). Register number must be aligned to the data size.
Data is transferred on EX2.
This instruction may be executed together with Shifter or Multiplier instructions. It can not be executed in parallel to ALU or accelerator instructions in the same compute unit.

~~1.1.2.~~3.1.2.TRs(d)(q) = Rm(d)(q)

The data in the register Rm is transferred to TRs. Data is single (32 bits), double (64 bits) or quad (128 bits). Register number must be aligned to the data size.
Data is transferred on EX2.
This instruction may be executed together with ALU instruction, Multiplier instruction or accelerator compute instruction. It can not be executed in parallel to shifter instruction in the same compute unit.

~~1.1.3.~~3.1.3.Rs(d)(q) = THRm(d)(q)

The data in the register THRm is transferred to Rs. Data is single (32 bits), double (64 bits) or quad (128 bits). Register number must be aligned to the data size.
Data is transferred on EX2.
This instruction may be executed together with Shifter or Multiplier instructions. It can not be executed in parallel to ALU or accelerator instructions in the same compute unit.
The quad option is not implemented because there are only two registers, it is reserved for future use.

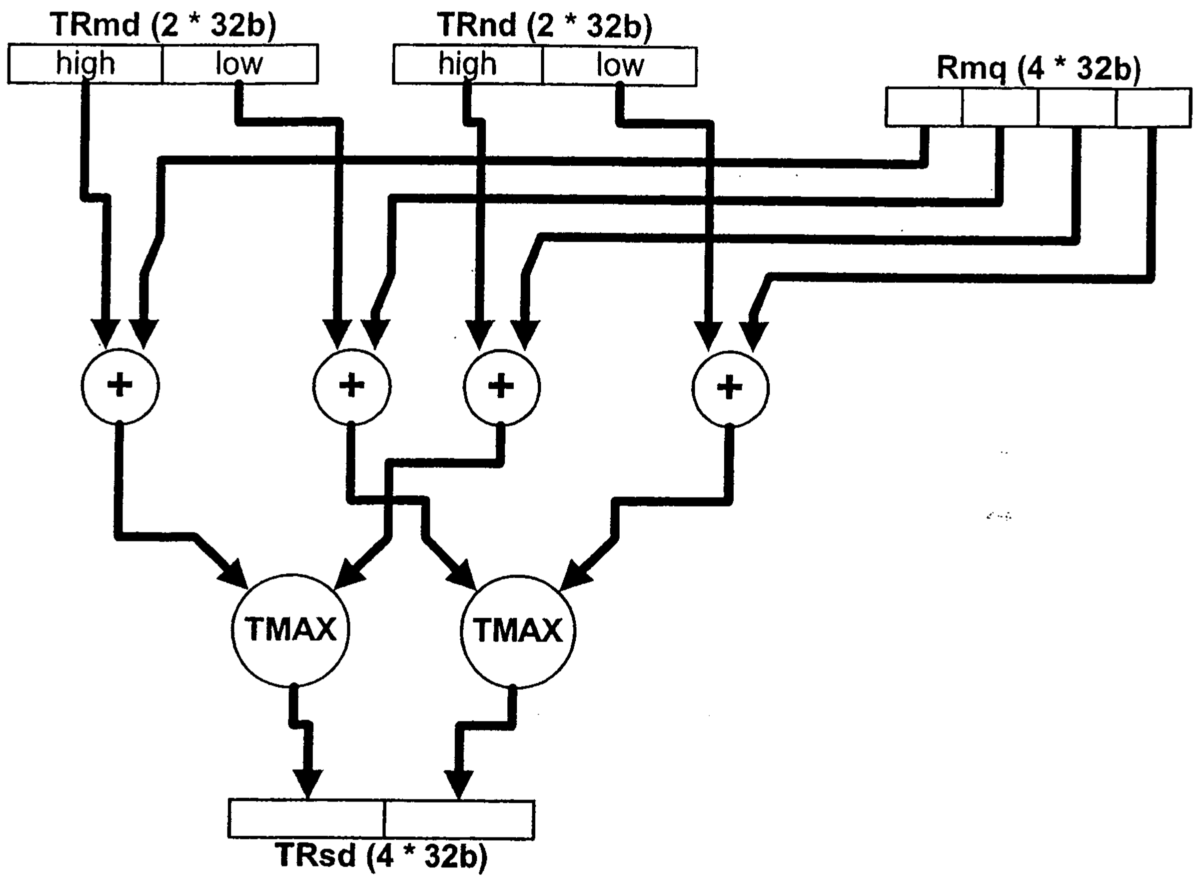
~~1.1.4.~~3.1.4.THRs(d)(q) = Rm(d)(q)

The data in the register Rm is transferred to THRs. Data is single (32 bits), double (64 bits) or quad (128 bits). Register number must be aligned to the data size.
Data is transferred on EX2.
This instruction may be executed together with ALU instruction, Multiplier instruction or accelerator compute instruction. It can not be executed in parallel to shifter instruction in the same compute unit.
The quad option is not implemented because there are only two registers, it is reserved for future use.

1.2.3.2.TMAX

1.1.1.3.2.1.TRsd = TMAX(TRmd + Rmq_h, TRnd + Rmq_l) ~~(init)~~

High part of Rmq is added to TRmd, low part of Rmq is added to TRnd, and TMAX function is executed between both add results



This instruction can be executed in parallel to shifter instructions, multiplier instructions and accelerator register load. It can not be executed in parallel to other accelerator instructions and ALU instructions of the same compute block.

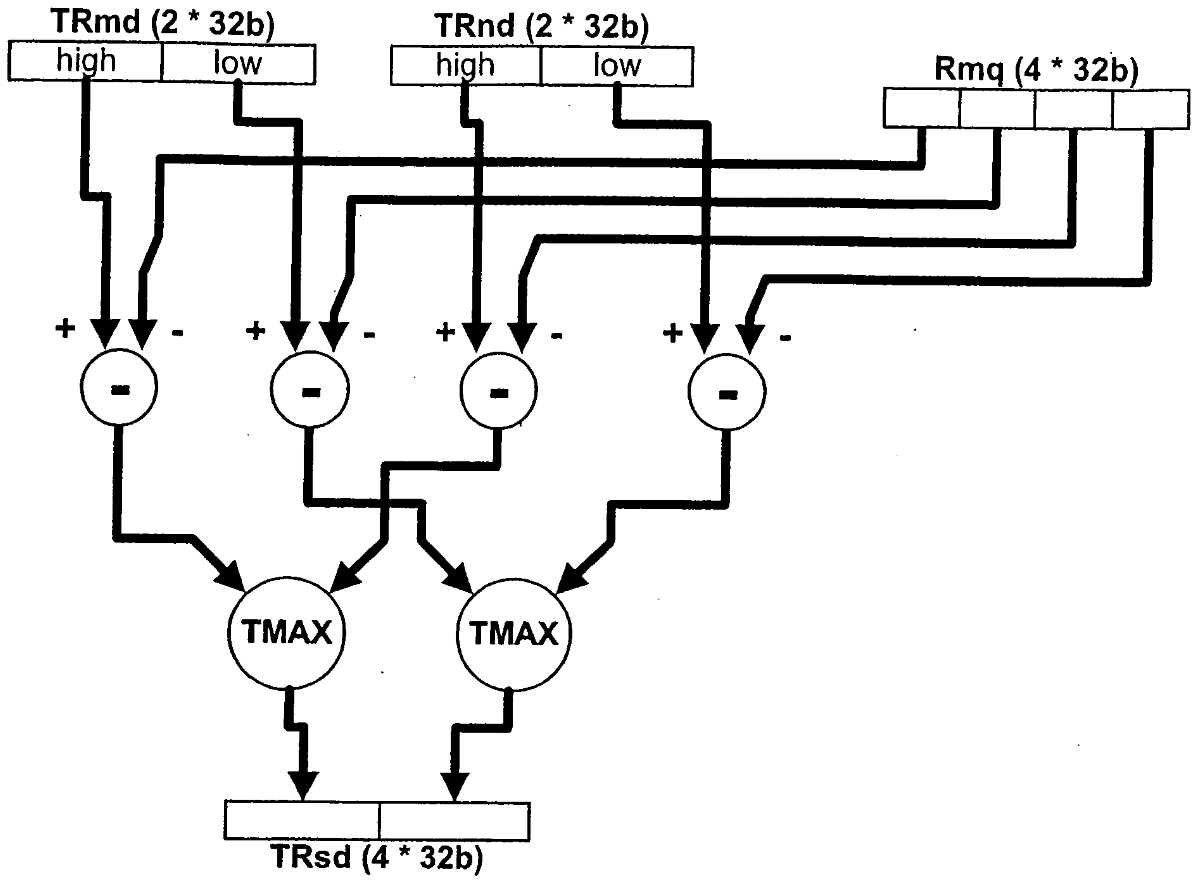
Affected flags – Overflow (TROV) is calculated every cycle

TRSPOV is set whenever a positive overflow occurs and cleared only by X/Ystat load

TRSNOV is set whenever a negative overflow occurs and cleared only by X/Ystat load

~~1.1.2.3.2.2.~~TRsd = TMAX(TRmd - Rmq_h, TRnd - Rmq_l) ~~(init)~~

High part of Rmq is subtracted from TRmd, low part of Rmq is subtracted from TRnd, and TMAX function is executed between both subtract results



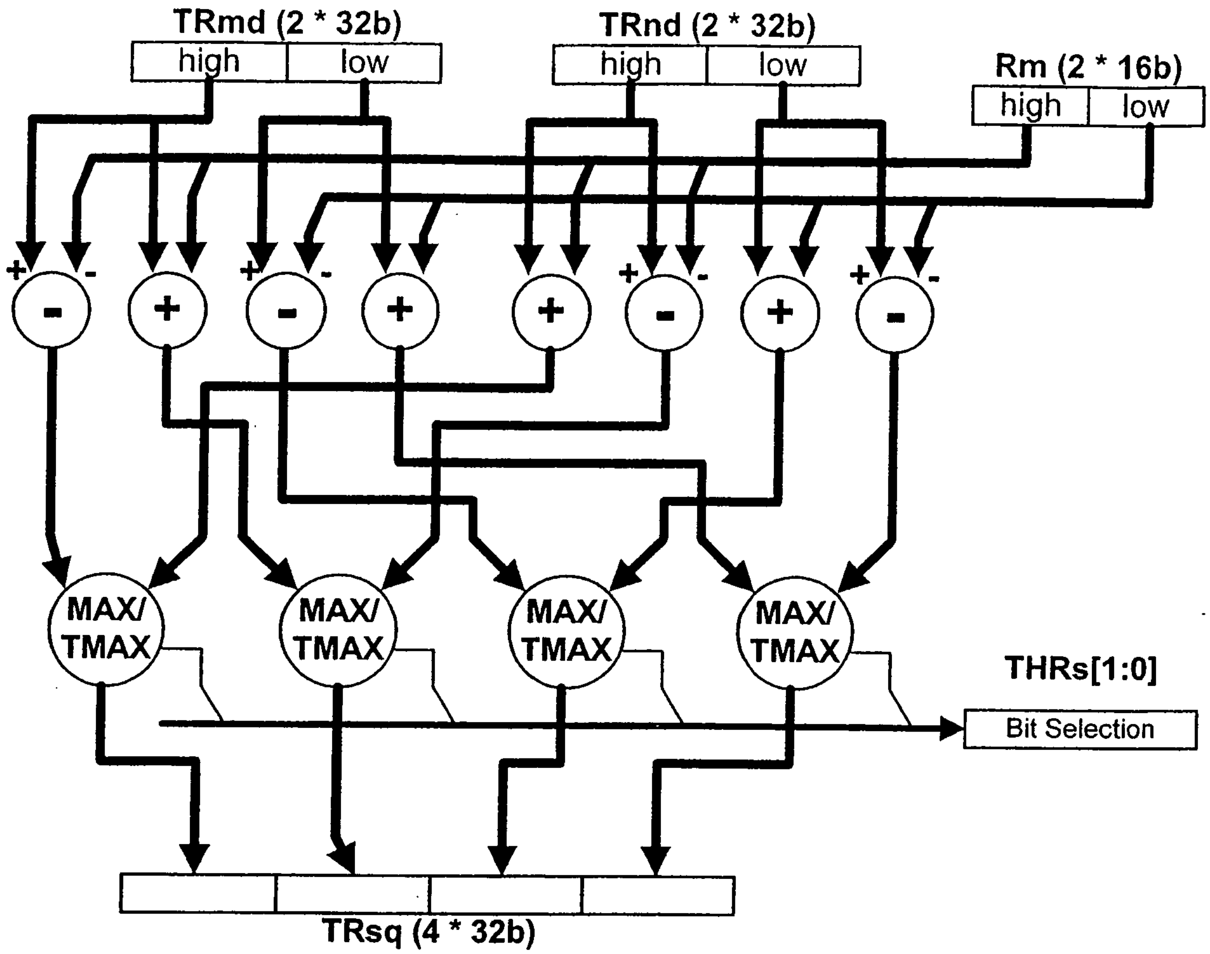
This instruction can be executed in parallel to shifter instructions, multiplier instructions and accelerator register load. It can not be executed in parallel to other accelerator instructions and ALU instructions of the same compute block.
Affected flags – Overflow (TROV) is calculated every cycle
TRSPOV is set whenever a positive overflow occurs and cleared only by X/Ystat load
TRSNOV is set whenever a negative overflow occurs and cleared only by X/Ystat load

ANALOG DEVICES

~~1.3.~~3.3.Add Compare Select

3.3.1. TRsq = ACS (TRmd, TRnd, Rm) (TMAX)

Rm is added to and subtracted from each word in TRmd and TRnd. The results of the add and subtract are compared in trellis order as shown below



Trellis history register THR[1:0] is updated with the selection of the MAX. After every ACS operation the four selection decisions are ~~shifted~~ loaded into bits 31:28 of register ~~the~~ THR1~~:0 register~~, while the content of THR1:0 is shifted right 4 bits~~-down~~.

Option TMAX replaces the MAX function with TMAX i.e. added to the value from table:
Ln(1+ e** (-Ia – bI))   where A and B are the two compared values

This instruction can be executed in parallel to shifter instructions, multiplier instructions and accelerator register load. It can not be executed in parallel to other accelerator instructions and ALU instructions of the same compute block.
Affected flags – Overflow (TROV) is calculated every cycle
TRSPOV is set whenever a positive overflow occurs and cleared only by X/Ystat load
TRSNOV is set whenever a negative overflow occurs and cleared only by X/Ystat load

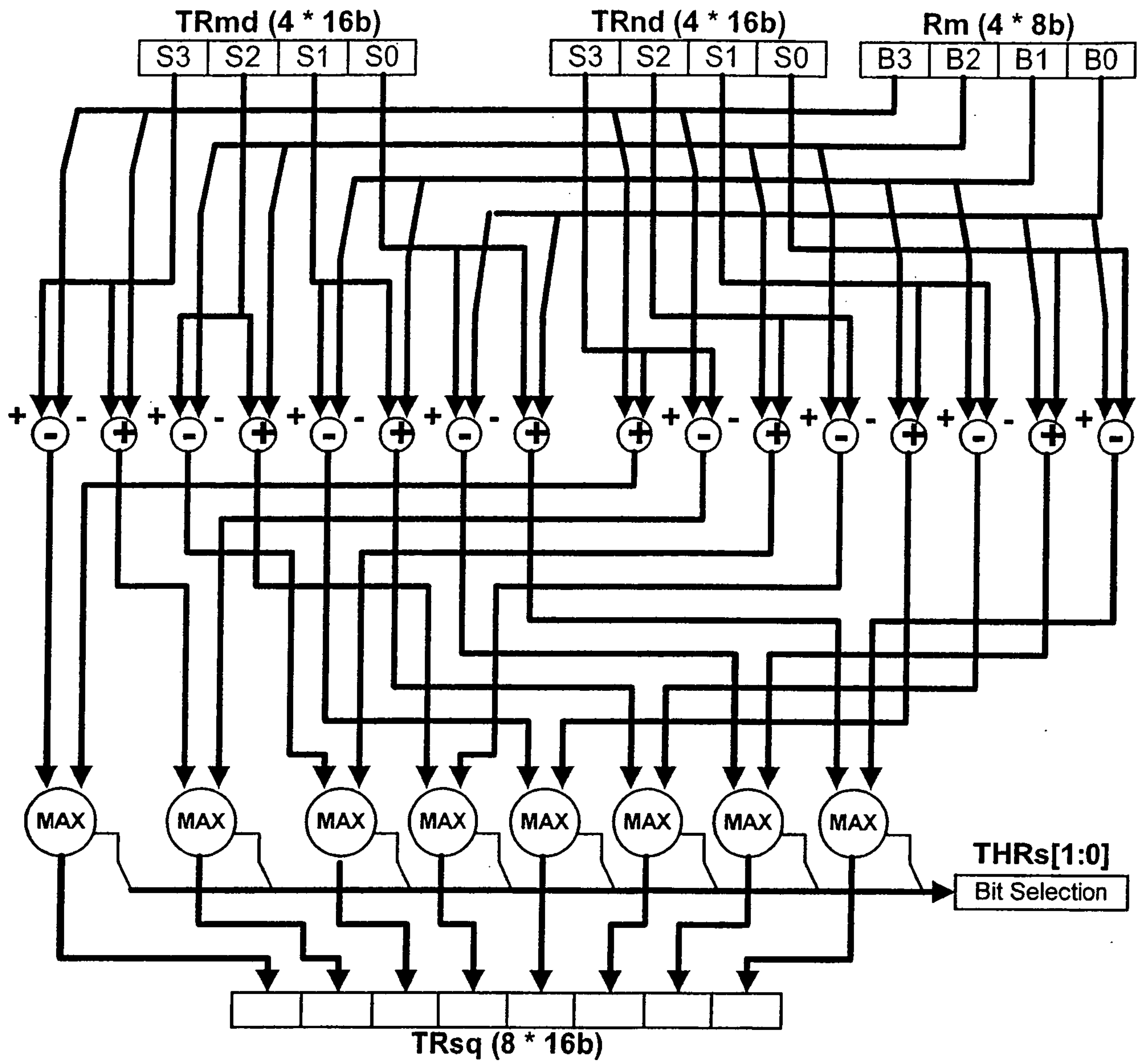
The way to use the function in trellis operation:

Loop: TR11:8 = ACS (TR1:0, TR5:4, R0);;
TR15:12 = ACS (TR3:2, TR7:6, R0);;
TR3:0 = ACS (TR9:8, TR13:12, R1);;
TR15:12 = ACS (TR11:10, TR15:14, R1); if nLC0E. jump Loop;;

~~1.1.2.~~3.3.2.STRsq = ACS (TRmd, TRnd, Rm)

This instruction is identical to regular ACS with two differences:

1. Calculates 4 butterflies of 16 bits instead of two 32 bit butterflies (input register Rm holds four 8 bit values).
2. Option TMAX is not supported here.



Trellis history register THR[1:0] is updated with the selection of the MAX. After every ACS operation the eight selection decisions are ~~shifted~~ loaded into bits 31:24 of register ~~the~~ THR1:~~0~~ register, while the content of THR1:0 is shifted right 8 bits ~~down~~.

This instruction can be executed in parallel to shifter instructions, multiplier instructions and accelerator register load. It can not be executed in parallel to other accelerator instructions and ALU instructions of the same compute block.

Affected flags – Overflow (TROV) is calculated every cycle

TRSPOV is set whenever a positive overflow occurs and cleared only by X/Ystat load

TRSNOV is set whenever a negative overflow occurs and cleared only by X/Ystat load

~~1.1.3.~~3.3.3.Rsq = TRaq, (S)TRsq = ACS (TRmd, TRnd, Rm) (TMAX)

This instruction is identical to previous – TRsq = ACS (TRmd, TRnd, Rm) (TMAX) with the addition of Saving Traq (quad accelerator register) in compute register file. Note that TRaq may not be identical to TRsq

This instruction can be executed in parallel to multiplier instructions and accelerator register load. It can not be executed in parallel to other accelerator instructions and ALU instructions of the same compute block.

ANALOG DEVICES

~~1.4.~~3.4.Permute

~~1.1.1.~~3.4.1.Rsd = Permute (Rmd, Rn)

The result – Rsd is composed of bytes from first operand Rmd, which are selected by control word Rn.
Rn is broken into 8 nibbles (4 bit fields), and Rsd is broken into 8 bytes. Each nibble in Rn is the control of the corresponding byte in Rmd (e.g. nibble 2 in Rn, which is bits 11:8, corresponds to byte 2 in Rsd, which is bits 23:16). The control word selects which byte in Rmd is written to the corresponding byte in Rsd. The decode of a nibble in Rn is:

0b0000:	Select byte 0 – bits 7:0 of low word
0b0001:	Select byte 1 – bits 15:8 of low word
0b0010:	Select byte 2 – bits 23:16 of low word
0b0011:	Select byte 3 – bits 31:24 of low word
0b0100:	Select byte 4 – bits 7:0 of high word
0b0101:	Select byte 5 – bits 15:8 of high word
0b0110:	Select byte 6 – bits 23:16 of high word
0b0111:	Select byte 7 – bits 31:24 of high word
0b1XXX:	Reserved

Different nibbles in Rn may point to the same byte in Rm. In this case the same byte in Rmd is duplicated in Rsd.
Affected flags: None

This instruction can be executed in parallel to shifter instructions, multiplier instructions and accelerator register load. It can not be executed in parallel to other accelerator instructions and ALU instructions of the same compute block. (TBD – change it to shifter instruction???)

Example:

R1:0 = 0x01234567_89abcdef
R4 = 0x03172454
R7:6 = permute (R1:0, R4)
Result R7:6 is 0xef89cd01_ab674567

~~1.1.2.3.4.2.~~Rsd = Permute (Rm, -Rm, Rn)

The result – Rsd is composed of bytes from first operand Rm, and it's ~~negated~~ two's compliment value -Rm, which are selected by control word Rn.
Rn is broken into 8 nibbles (4 bit fields), and Rsd is broken into 8 bytes. Each nibble in Rn is the control of the corresponding byte in Rmd (e.g. nibble 2 in Rn, which is bits 11:8, corresponds to byte 2 in Rsd, which is bits 23:16). The control word selects which byte in Rmd is written to the corresponding byte in Rsd. The decode of a nibble in Rn is:

0b0000:	Select byte 0 – bits 7:0 of Rm
0b0001:	Select byte 1 – bits 15:8 of Rm
0b0010:	Select byte 2 – bits 23:16 of Rm
0b0011:	Select byte 3 – bits 31:24 of Rm
0b0100:	Select byte 4 – bits 7:0 of -Rm
0b0101:	Select byte 5 – bits 15:8 of -Rm
0b0110:	Select byte 6 – bits 23:16 of -Rm
0b0111:	Select byte 7 – bits 31:24 of -Rm
0b1XXX:	Reserved

Different nibbles in Rn may point to the same byte in Rm. In this case the same byte in Rmd is duplicated in Rsd.

This instruction can be executed in parallel to shifter instructions, multiplier instructions and accelerator register load. It can not be executed in parallel to other accelerator instructions and ALU instructions of the same compute block. (TBD – change it to shifter instruction???)

Affected flags – Overflow (TROV) is calculated every cycle, set if Rm is maximum negative
TRSPOV is set whenever a positive overflow occurs and cleared only by X/Ystat load~~Affected flags: None~~

Example:
R0 = 0x01234567
-R0 = 0xFEDCBA99
R4 = 0x03172454
R7:6 = permute (R0, -R0, R4)
Result R7:6 is 0x670145FE_2399BA99

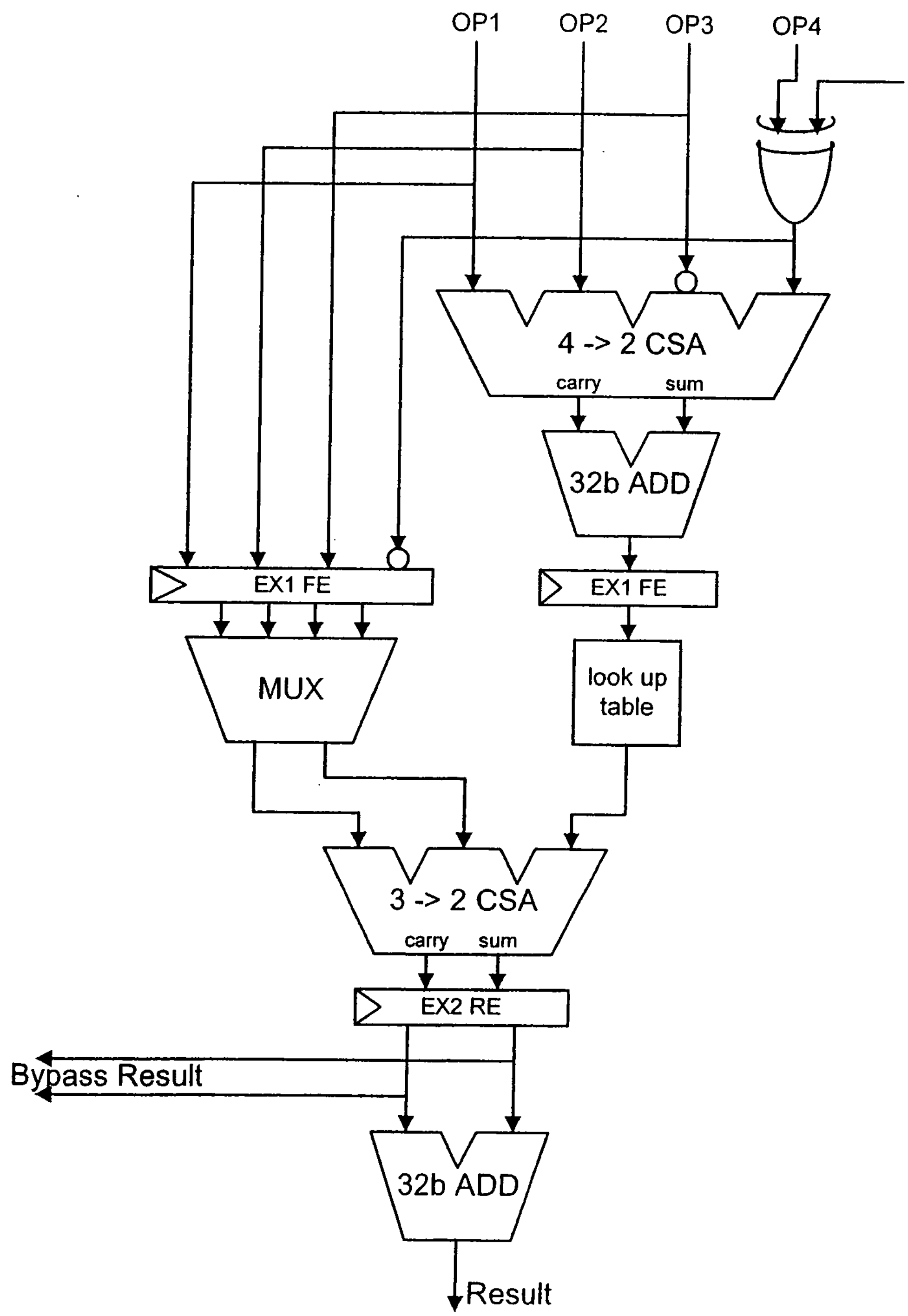
[redacted]

4. Instruction Flow

All the instructions are executed in the compute pipeline. The pipeline stages are:
Decode stage: instruction is received in the compute register file. RF identifies it as an accelerator instruction.
Integer stage: Dependency check is done in RF (for compute registers) and in accelerator (for accelerator registers)
Access stage: Operands are read from register files
EX1 stage: calculation begins
EX2 stage: calculation is completed and the result is written into the destination registers
All instructions have a dependency check. Every use of a result of the previous line causes a stall for one cycle. An exception is the instruction ACS, which can use previous result of ACS instruction as TRmd or TRnd with no stall (TBD if on either operands or(TBD) just first operand).
Data transfer from accelerator register to compute register file has no dependency – The data transfer is executed in EX2.
All instructions may be aborted either by ABORT (caused by regret of speculative flow or by exception) or by a false condition resolution, when the instruction is predicated (CC bit is set).
The accelerator register load can be executed in parallel to other accelerator instructions. Its code is a shifter instructions, while the other instructions are coded as ALU instructions.

5. Implementation

5.1. TMAX and ACS instructions



The implementation of TMAX and ACS instructions execution is shown in the figure above. This datapath is duplicated 4 times in order to execute four calculations in parallel. On the first phase the first CSA and the full adder compare the two elements that should be compared. For ACS instruction it should calculate:
OUT_ex1_high = (TRm – Rm) – (TRn + Rm) = TRm – TRn – (2 * Rm) for word 0 and 2. And
OUT_ex1_high = (TRm + Rm) – (TRn - Rm) = TRm – TRn + (2 * Rm) for word 1 and 3
In this case only three inputs are used. The fourth is used in case of dependency with a previous ACS instruction. In this case the sum and carry are taken from the second stage of the pipeline.
In TMAX instruction the output of the first stage is:
OUT_ex1_high = (TRm + Rm_high) – (TRn + Rm_low)
The sign of OUT_ex1_high is used for selection between the two inputs of TMAX function. Them result is used as input of the TMAX table.
On clock low of EX1 the inputs are selected according to the sign of OUT_ex1_high, and the TMAX table drives drives its output. The TMAX table output and the two inputs selected from the input are summed in the second CSA (CSA 3->2). The inputs of the CSA are:

In ACS word 0 and 2:

If OUT_ex1_high > 0:	OUT_ex1_low = TRm – Rm
Else:	OUT_ex1_low = TRn + Rm

In ACS word 1 and 3:

If OUT_ex1_high > 0:	OUT_ex1_low = TRm + Rm
Else:	OUT_ex1_low = TRn - Rm

In TMAX:

If OUT_ex1_high > 0:	OUT_ex1_low = TRm – Rm_high
Else:	OUT_ex1_low = TRn – Rm_low

The result of the CSA is redundant i.e. two vectors OUT_s_ex1_low and OUT_c_ex1_low. The final result is the real sum of these two vectors. These two vectors are used for bypass if the next instruction uses this instructions result.
The full sum is calculated on EX2 clock high.

~~1.2.5.2.~~TMAX table

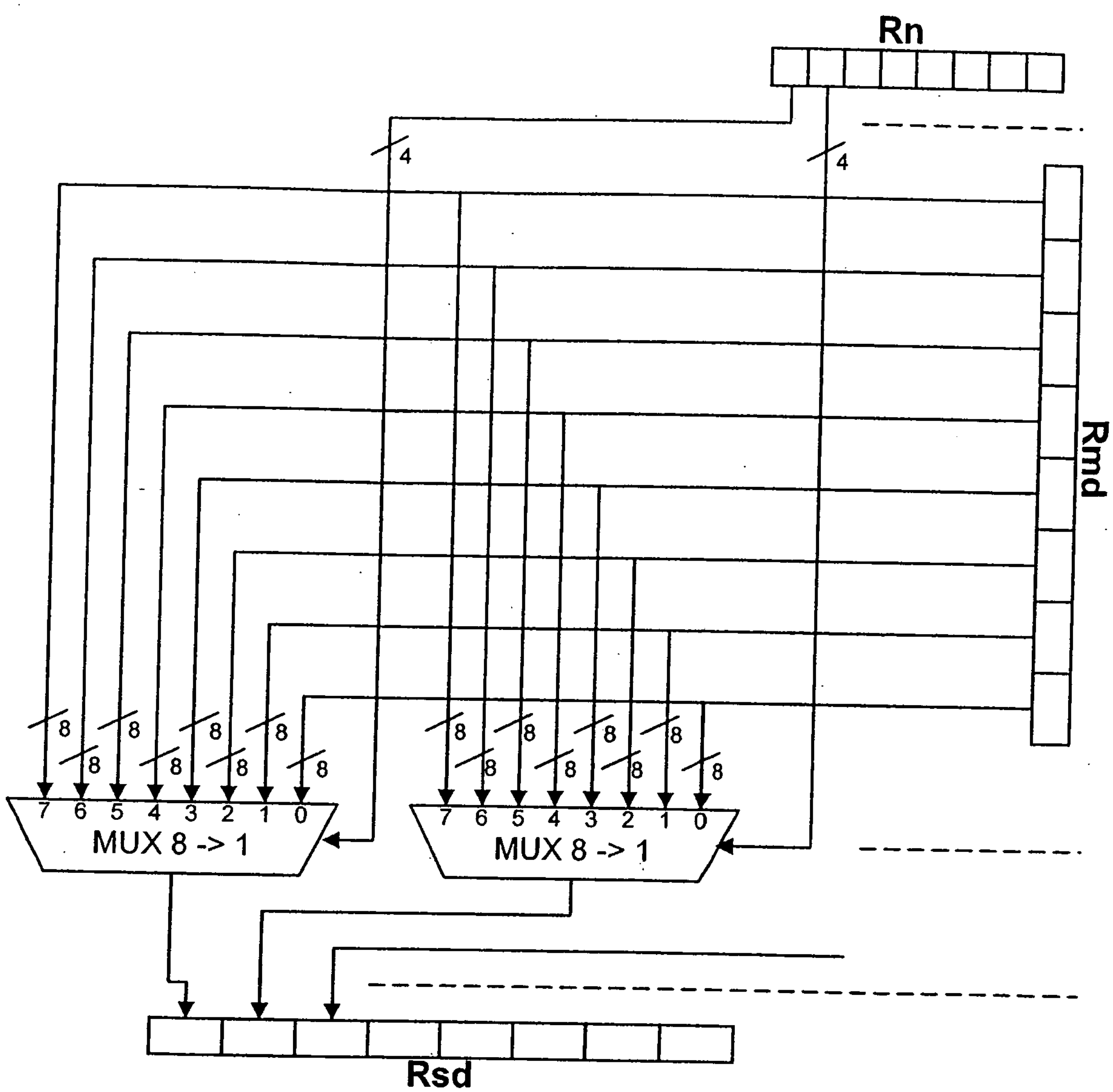
The table input is the result of the subtract on clock high of pipe stage EX1. If the result is negative, it is inverted (assuming that difference between one's compliment and two's compliment is within the allowed error). The input to the table is the 7 LSB's of the compare subtract result. If the compare subtract result is larger than 7 bits, the output is zero.

Input (binary)	Negative input	Output (Hex)	Error
00..0000000X	11..1111111X	56	+/-2
00..0000001X	11..1111110X	52	+/-2
00..0000010X	11..1111101X	4F	+/-2
00..0000011X	11..1111100X	4B	+/-2
00..0000100X	11..1111011X	47	+/-2
00..0000101X	11..1111010X	44	+/-2
00..000011XX	11..111100XX	3F	+/-3
00..000100XX	11..111011XX	39	+/-3
00..000101XX	11..111010XX	34	+/-3
00..000110XX	11..111001XX	2F	+/-3
00..000111XX	11..111000XX	2A	+/-2
00..001000XX	11..110111XX	25	+/-3
00..001001XX	11..110110XX	22	+/-2
00..001010XX	11..110101XX	1E	+/-2
00..001011XX	11..110100XX	1B	+/-2
00..00110XXX	11..11001XXX	17	+/-3
00..00111XXX	11..11000XXX	12	+/-2
00..0100XXXX	11..1011XXXX	0C	+/-2
00..0101XXXX	11..1010XXXX	08	+/-2
00..011XXXXX	11..100XXXXX	04	+/-2
>00..100000000	<11..01111111	00	+/-2

The table above shows an option for table with output error of up to 3 * LSB. This table has 21 entries. The table below shows an option for table with output error of up to 8 * LSB. This table has 9 entries.

Input (binary)	Negative input	Output (Hex)	Error
00..00000XXX	11..11111XXX	50	+/-8
00..00001XXX	11..11110XXX	42	+/-7
00..00010XXX	11..11101XXX	38	+/-6
00..00011XXX	11..11100XXX	2D	+/-4
00..0010XXXX	11..1101XXXX	21	+/-7
00.. 0011XXXX	11.. 1100XXXX	15	+/-5
00..010XXXXX	11..101XXXXX	0B	+/-5
00..011XXXXX	11..100XXXXX	04	+/-2
>00..100000000	>11..011111111	00	+/-2

~~1.3.5.3.~~Permute Instruction



~~1.4.5.4.~~Floor plan

The accelerator is connected to all compute operand and result busses.
It should be placed between multiplier and shifter.

6. Open Issues

1. The following things are defined here as superset, but should be examined in the future if will be supported:

ANALOG DEVICES

- Short (16 bits) support for ACS
- Instruction TRsd = TMAX(TRmd - Rmq_h, TRnd - Rmq_l)
- Instruction Rsd = Permute (Rm, -Rm, Rn)
- Bypass for one or two operands (in order to support one cycle throughput) on ACS instruction

2. Which (if at all) exceptions are supported by accelerator?
3. Rsd = Permute (Rm, -Rm, Rn) – should it be single or double?
4. Can the permute be in the shifter instruction group? The purpose is to enable execution of permute in parallel to ACS, TMAX and ALU instructions.
5. TMAX table – the assumption is that the 21 entry table will be used (+/-3*LSB error).